Introduction to Machine Learning for the Life Sciences Lab Sessions

by

John Joseph Valletta
University of Exeter, UK

April 7, 2015

1 Introduction

The aim of these lab sessions is to get you started with applying machine learning algorithms to various types of datasets. The focus is to showcase the verstaility of these methods and give you a flavour of the sort of problems that can be tackled. We will not be spending time on the topic of *feature extraction*, that is, how to extract features which have the best predictive capabilities. This is usually the crux of any predictive modelling problem but which is domain-specific.

We will be using the programming language **R**, mainly because most life scientists are already familiar with it, but also because it is an open-source platform that gives you access to state-of-the-art algorithms. **Python** (also free and extremely popular with computer scientists) and **MATLAB** ® are also popular choices. You can also find faster implementations in **C/C++**.

These lab sessions are not intended to be an **R** tutorial, so if you want to learn more about **R**, please refer to the myriad of free resources available on the web.

Installing R: http://www.r-project.org/

Installing RStudio: http://www.rstudio.com/

> It is imperative that you read and familiarise yourself with the documentation of any machine learning algorithm before using them

Everyone might have different packages available in their **R** installation, so before calling `library(Package Name)`, please run `install.packages("Package Name")`, where Package Name is the name of the package you need e.g `mclust`

2 Getting started using an artificial dataset

Data:	artificial data from a 3D multivariate gaussian distribution
Task:	cluster data points (unsupervised)
Method:	k-means and Gaussian mixture models

Before starting using real datasets, let us apply some of the algorithms we discussed to an artificial dataset that we can visualise and control. Let us generate 5 sets of data from a 3-dimensional gaussian distribution and visualise the data.

```
library(MASS) # mvrnorm (multivariate normal)
library(rgl) # plot3d
N <- 100 # number of data points
sigma2 <- 1 # variance of data points
covMatrix <- sigma2*diag(3) # assume the same variance in all three directions
groupA <- mvrnorm(n=N, mu=c(3, 3, 9), Sigma=covMatrix)
groupB <- mvrnorm(n=N, mu=c(3, 9, 3), Sigma=covMatrix)
groupC <- mvrnorm(n=N, mu=c(9, 9, 9), Sigma=covMatrix)
groupD <- mvrnorm(n=N, mu=c(9, 3, 3), Sigma=covMatrix)
groupE <- mvrnorm(n=N, mu=c(6, 6, 6), Sigma=covMatrix)
xTrain <- rbind(groupA, groupB, groupC, groupD, groupE) # training dataset is 5*N x 3
plot3d(xTrain, col="grey", xlab="x", ylab="y", zlab="z")
```

Visually you can probably deduce that there seem to be 5 different clusters. In practice however this distinction is unclear and we cannot visualise the data. Recall the main objective of clustering:

- High intra-cluster similarity
- Low inter-cluster similarity

To deduce the right number of clusters we can run k-means several times and plot the resultant intra- and inter-cluster sum-of-squares as a function of k

```
# k-means, let k vary from 2 to 10
kRange <-  seq(from=2, to=10, by=1)
intraClustSS <- rep(NA, length(kRange))
interClustSS <- rep(NA, length(kRange))
for (k in kRange){
    fit <- kmeans(x=xTrain, centers=k)
    intraClustSS[k-1] <- fit$tot.withinss # it's (k-1) because k starts from 2
    interClustSS[k-1] <- fit$betweenss
}
minY <- min(c(intraClustSS, interClustSS))
maxY <- max(c(intraClustSS, interClustSS))
plot(kRange, interClustSS, type="o", pch=1, col="blue", lwd=3, lty=1, ylim=c(minY, maxY)
    , xlab="k")
points(kRange, intraClustSS, type="o", pch=1, col="red", lwd=3)
legend("topleft", c("inter-cluster sum-of-squares", "intra-cluster sum-of-squares"), bty
    ="n",
```

```
       col=c("blue", "red"), pch=1, lwd=3, lty=1)
```

After $k = 5$, the sum-of-squares doesn't change that much, so it can be deduced that there are 5 distinct clusters in this dataset. Let us visualise the result:

```
fit <- kmeans(x=xTrain, centers=5)
plot3d(xTrain, col=fit$cluster, xlab="x", ylab="y", zlab="z")
```

Question(s):

1. What happens to the inter/intra-cluster sum-of-squares plot as we increase `sigma2`?

Let us perform a similar analysis but using a mixture of Gaussian models, where we will fit k Gaussians. This time, instead of the inter/intra-cluster sum-of-squares we will plot the BIC (Bayesian Information Criterion), a goodness-of-fit measure that penalises model complexity (k). The `Mclust` function uses BIC to decide which model to reatin. To speed up the process we will also tell `Mclust` that we want a Gaussian with equal variances in all directions (`modelNames="EII"`).

```
library(mclust) # Gaussian mixture models
fit <- Mclust(data=xTrain, G=seq(10), modelNames="EII")
summary(fit)
plot(fit$BIC, type="o", pch=1, col="blue", lwd=3, lty=1, xlab="k")
legend("topleft", "BIC (Bayesian Information Criterion)", bty="n",
       col=c("blue", "red"), pch=1, lwd=3, lty=1)
plot(fit, what="density", lwd=2)
```

Question(s):

1. Try the same but without specifying `modelNames`. Which model does it retain?
2. How does the BIC plot and type of model retained change when increasing `sigma2`?

3 Clustering gene expression data

Data:	gene expression (data of T- and B-cell Acute Lymphocytic Leukemia)
Task:	deduce the number of distinct phenotypes (unsupervised)
Method:	agglomerative hierarchical clustering

Agglomerative hierarchical clustering is a popular choice in gene expression studies. The data comes as a $N \times D$ matrix, where N is the number of genes and D is the number of biological samples (e.g patients with different medical conditions or different tissue samples). The numerical value is a measure of how much the nth gene is expressed for the dth biological sample. There are usually two key questions that can be addressed with clustering:

1. Which biological samples exhibit a similar gene expression *signature*?
2. Which genes vary in a similar fashion?

Here we will use the popular acute lymphoblastic leukemia (ALL) dataset. It is a 12625 (genes) $\times$ 128 (patients) microarray dataset. The patients were diagnosed with either a B- or T-cell acute lymphocytic leukemia. Because we have access to the labels (type and stage of the disease e.g B2) we can easily assess how well the clustering algorithm is doing as we expect the B's and T's to cluster together. Note however that availability of labels is not always the case and that the clustering algorithm does *not* know what the labels are. It is an unsupervised problem (we will turn it into a supervised problem later on).

The ALL dataset comes in the form of an `exprSet` object, a data structure used in Bioconductor. Bioconductor is an ensemble of **R** packages specifically written to analyse high-throughput genomic data. Let us install Bioconductor (for those who have not got it installed already) and extract the $N \times D$ gene expression matrix of interest [1].

```
source("http://bioconductor.org/biocLite.R") # Install Bioconductor
biocLite() # Install core packages (will take a few mins)
biocLite("ALL") # Install the ALL package
library(ALL) # Load the ALL package
data(ALL) # Loads ALL dataset to workspace
xTrain <- exprs(ALL) # Extract the 12625 x 128 dataset
colnames(xTrain) <- ALL$BT # Replace patient ID by disease stage to assess clustering
```

For the purpose of this exercise, let us just focus on clustering the 128 patients:

```
distance <- dist(as.matrix(t(xTrain), method="euclidean")) # Distance between patients
# Perform agglomerative hierarchical clustering using the "complete" linkage function
```

[1] Do not worry too much about the details the objective is just to extract the data, but if your work involves this type of data it is recommended that you familiarise yourself with Bioconductor

```r
fit <- hclust(distance, method="complete")
plot(fit, cex=0.5) # Visualise the resultant dendrogram
```

Question(s):

1. Keep the distance method fixed and change the linkage method (e.g single, average). What happens to the dendrogram?
2. Keep the linkage method fixed and change the distance method (e.g manhattan, minkowski). What happens to the dendrogram?
3. Try using a correlation distance metric and repeat the above. Hint:

```r
distmethod <- function(x) as.dist(1-cor(x))
distance <- distmethod(xTrain)
```

4 Species distribution modelling

Data:	*Bradypus variegatus* (brown-throated slot) geographic distribution data from Phillips et al. (2006)
Task:	produce a map showing the likely geographic distribution of *Bradypus variegatus* (unsupervised)
Method:	Gaussian mixture models

An ubiquitous challenge in conservation biology is to produce species distribution maps. The objective is to show how in different regions there is more or less chance of observing the species under consideration. The dataset consists of 116 latitude and longitude recordings of where *Bradypus variegatus* was observed. This is a classic density estimation problem, which in this case is equivalent to a 2D histogram. Let us start by retrieving the dataset which is found in the `dismo` package and plot a point for every observation.

```
library(raster) # functions for gridded spatial data
library(dismo) # package containing the Bradypus variegatus dataset
library(rworldmap) # access to map of the world
dataPath <- file.path(system.file(package="dismo"), "ex") # path to data location
bradypusFilePath <- file.path(dataPath, "bradypus.csv") # path to Bradypus dataset
data <-  read.table(file=bradypusFilePath, header=T, sep=",", skip=0) # read data
worldMap <- getMap(resolution = "low") # access world map, xlim = long, ylim = lat
plot(worldMap, xlim = c(-85, -40), ylim = c(-25, 20), asp = 1, axes=T)
points(data[, 2], data[, 3], pch=20, col="blue", cex=0.5) # add observed data points
```

The question now is which variables should we use to fit this histogram. A naïve approach is to use the observed latitude and longitude, after all we would expect to find the same species within short geographical distances from where we have already observed some. This approach however ignores the habitat properties inhabited by *Bradypus variegatus*. A better method would be to use environmental variables (e.g temperature and precipitation) which characterise the species' habitat to fit this distribution and then convert back to the latitude/longitude domain. The `dismo` package contains a number of bioclimatic variables stored as GRD files. This is a popular file format used in GIS (Geographic Information System) which contains grid cell values for that variable. The naming convention for these variables can be found here [2]. The data resolution is $0.5° \times 0.5°$ which is sufficient for illustrative purposes. The steps taken in this task look something like this:

[2]Note that temperature data is stored as °C $\times 10$ as discussed here

1. Read in and plot environmental variables which cover the geographical area of interest.

```
grdFiles <- list.files(path=dataPath, pattern='grd', full.names=T) # read path of
    all .grd files
envData <- stack(grdFiles) # stack all environmental variables
plot(envData) # plot to confirm data is OK
```

2. For every observed data point, i.e 116 lat/long positions, extract a value for the environmental variable.
   i.e (temperature, precipitation...)$=f$(latitude, longitude)

```
xTrain <- extract(envData, data[, 2:3])
```

3. Standardise the environmental data as ranges vary widely.
   i.e $x_{\text{standardise}} = \frac{x-\mu}{\sigma}$
   This is a very important step and if omitted the algorithm might fail to converge

```
meanXTrain <- apply(xTrain, 2, mean) # compute mean of every column
sdXTrain <- apply(xTrain, 2, sd) # compute sd of every column
xTrain <- sweep(xTrain, 2, meanXTrain, FUN="-") # x - mean
xTrain <- sweep(xTrain, 2, sdXTrain, FUN="/") # (x - mean)/sigma
```

4. Fit a distribution to the standardised data using Gaussian mixture models. Instead of using all 9 variables however, only a subset of 3 are considered. The reasons are twofold: i) some variables are highly correlated with each other (e.g mean annual temperature and maximum temperature of warmest month), but most importantly because of ii) the *curse of dimensionality*. The more variables (dimensions) are considered the more the space will be sparsely populated, unless dataset is extremely large.

```
library(mclust)
envVariables <- c("bio1","bio12", "biome") # chosen environmental variables
fit <- densityMclust(data=xTrain[, envVariables]) # fit distribution
```

5. Compute the value of the fitted distribution for every geographical grid of interest and plot the species geographical distribution map.

```
# Standardise test data (xTest is the whole grid not just the 116 obs.)
xTest <- as.data.frame(subset(envData, subset=envVariables))
xTest <- sweep(xTest, 2, meanXTrain[envVariables], FUN="-")
xTest <- sweep(xTest, 2, sdXTrain[envVariables], FUN="/")
# Read the distribution at the test points
notNA <- complete.cases(xTest) # ignore sea points (NA = sea)
pred <- rep(NA, dim(xTest)[1])
pred[notNA] <- predict(fit, newdata=xTest[notNA, ])
# Create a data frame with values for the distribution for every grid cell
df <- data.frame(pred=as.matrix(pred, nrow=nrow(envData), ncol=ncol(envData)))
coordinates(df) <- coordinates(envData) # set spatial coordinates
gridded(df) <- TRUE # coerce to SpatialPixelsDataFrame
rasterDF <- raster(df) # coerce to raster
```

```
library(RColorBrewer) # package for pretty colour palettes
plot(rasterDF, col=brewer.pal(n=9, name="Reds"))
points(data[, 2], data[, 3], pch=20, col="blue", cex=0.5) # add observations
```

Question(s):

1. Compare your geographical distribution map with Phillips et al. (2006) pg. 252. Do you think that a Gaussian mixture model is appropriate for this problem?

2. Pick different combinations of `envVariables` and describe how the species distribution map changes. Try also single environmental variables and see what happens.

5 Predicting forest cover type from cartographic attributes

Data:	forest cover type
Task:	predict the forest cover type given a set of cartographic measurements (supervised)
Method:	decision trees and random forests

This problem appeared as a competition on Kaggle. It is a multi-class problem where several cartographic attributes are used to predict the forest cover type. Data comes from the Roosevelt National Forest of northern Colorado and each row represents an observation over a 30m × 30m patch. The seven distinct forest cover types (`Cover Type`) are:

1. Spruce/Fir
2. Lodgepole Pine
3. Ponderosa Pine
4. Cottonwood/Willow
5. Aspen
6. Douglas-fir
7. Krummholz

The cartographic attributes are:

`Elevation`
Elevation in meters

`Aspect`
Aspect in degrees azimuth

`Slope`
Slope in degrees

`Horizontal Distance To Hydrology`
Horizontal distance to nearest surface water features

`Vertical Distance To Hydrology`
Vertical distance to nearest surface water features

`Horizontal Distance To Roadways`
Horizontal distance to nearest roadway

`Hillshade 9am`[3]
Hillshade index at 9am, summer solstice

[3]Hillshade index takes an integer value from 0 (complete shadow) to 255 (full sunlight)

`Hillshade Noon`
Hillshade index at noon, summer solstice

`Hillshade 3pm`
Hillshade index at 3pm, summer solstice

`Horizontal Distance To Fire Points`
Horizontal distance to nearest wildfire ignition points

`Wilderness Area`
4 binary columns (0 = absence or 1 = presence) of wilderness area designation

`Soil Type`
40 binary columns (0 = absence or 1 = presence) of soil type designation

For infortmation on wilderness area designation and soil type please check additional information, however it is not essential for the purpose of this exercise. Let us start by reading the dataset in and split it into a training and testing dataset.

ADD A NOTE TO LOCATE WHERE THE DATA FILE IS ONCE I DECIDE WHERE TO PUT IT
MAYBE PUT IT ON GITHUB THEN JUST DO A GET FROM THERE...

```
data <- read.table("ForestCoverData.csv", header=T, sep=",", skip=0) # Takes a few secs
data <- data[complete.cases(data), ] # Only 2 cases of missing data so ignore
data$Cover_Type <- factor(data$Cover_Type) # Convert to a factor
NTOTAL <- dim(data)[1] # Total number of observations
NTRAIN <- 12000 # Let us use this many training datapoints and the rest for testing
set.seed(101) # Just so we can reproduce results
trainIndices <- sample(x=NTOTAL, size=NTRAIN, replace=FALSE)
```

Let us start by fitting a simple decision tree to the data.

```
library(rpart)
library(rattle) # For fancyRpartPlot to plot rpart tree nicely
fit <- rpart(Cover_Type ~ ., data=data, subset=trainIndices, method="class")
fancyRpartPlot(fit) # Plot decision tree
predClass = predict(fit, type="class") # Pred class on training dataset
confusionMatrix <- table(data$Cover_Type[trainIndices], predClass)
classError <- 1 - diag(prop.table(confusionMatrix, 1)) # Compute misclassification rates
```

Result not promising, especially for `Cover Type` which are not 1, 2 or 3. Let us use a Random Forest and see how the `classError` changes.

```
library(randomForest)
fit <- randomForest(Cover_Type ~ ., data=data, subset=trainIndices, ntree=200,
    importance=TRUE) # Takes a few secs
fit$confusion # Confusion matrix
```

Misclassification rate has improved, but for some `Cover Type` this is still unsatisfactory, why? A quick look at the confusion matrix reveals how our training dataset is *unbalanced*. `Cover Type` 1 and 2 comprise 85% of the training examples. Having one class more prevalent in a dataset is a common problem in classification, for example, you may have lots of data for healthy people but only a few for those suffering a particular disease. There are two approaches to circumvent such problem:

1. **On a data level**: Data is balanced by under/over-sampling the original dataset. The aim is to get roughly equal number of classes in the training dataset
2. **On an algorithm level**: This depends on the algorithm used, but usually involves adjusting some objective function or specifying directly the class priors

Question(s):

1. Choose a training dataset that have roughly equal number of classes and repeat the fitting process (both a simple decision tree and Random Forest). What is the model's performance now? (compute misclassification error rates for both *training* and *testing* datasets. Hint: In **R** you can use the notation `-trainIndices` to access the testing indices.
2. Repeat the above, but instead of balancing the dataset, let us specify the class priors by setting the `classwt` in the `randomForest` function. How does the model's performance compare with the above?
3. Using the model which gave you the best performance, can you elucidate on the most important variables? Hint: Plot the variable importance plot `varImpPlot(fit)`
4. Instead of having 40 binary columns for `Soil Type` how else can you "code" this variable?

6 Classifying patients by their gene expression signature

Data: gene expression (data of T- and B-cell Acute Lymphocytic Leukemia)

Task: predict a patient's phenotype using their gene expression signature (supervised)

Method: decision trees

Here we will consider again the gene expression data from the previous clustering exercise. This time however we formulate it as a supervised learning problem. Given a gene expression signature can we predict the patient's phenotype? Let us start by considering only a binary classifier, that is, discriminating between B- and T-cell cancer types. As this is an illsutrative example and the sample size is fairly small let us use all available data for training[4].

```
library(ALL) # Load the ALL package
data(ALL) # Loads ALL dataset to workspace
xTrain <- t(exprs(ALL)) # Extract the 128 x 12625 dataset
yTrain <- factor(substr(ALL$BT,1,1)) # Class is either B or T
df <- data.frame(x=xTrain, y=yTrain) # Create a data frame, x - all inputs, y - class
```

Question(s):

1. Try logistic regression, a very popular statistical binary classifier (Hint: `glm(y ~., data=df, family=binomial(link="logit"))`). What's the outcome and why?
2. Use decision trees now (Hint: use the same code as listed in the previous Forest Cover Type exercise). What's the classifier's performance on the training dataset?
3. Repeat the above but now we want to infer the stage of the disease as well (Hint: Set `yTrain <-factor(ALL$BT)`)

[4]Note that if we do this, we cannot infer the generalisation error. In this case using cross-validation would have been a more sensible option, but this was not discussed in detail in the workshop yet